INTELLIGENT PCI BRIDGING

BACKGROUND

Many computer systems rely upon expansion busses to add functionality to the overall system. Generally, the added functionality takes the form of small printed circuit boards or other types of 'cards' that have on them the necessary components to allow the main processor to communicate with other devices. For example, video cards, audio cards, and network interface cards all provide added functionality to the system. The cards may communicate over an expansion bus, rather than being included on the main system bus.

Expansion busses are generally one of two types, an ISA (Industry Standard Architecture) bus or a PCI (Peripheral Component Interconnect) bus. The ISA standard was used initially, but became a bottleneck as processor speeds increased. Typically, most computer systems now employ PCI busses, or busses similar to PCI-X (PCI eXtended). The device that connects the PCI bus to the main system bus is usually referred to as a PCI bridge.

Expansion cards communicate with the CPU across the expansion bus. When the CPU needs an expansion device, such as a network interface card, to transmit data, it sets up a transmit ring in the memory to direct the device to the data to be transmitted by writing data descriptors to the transmit ring. The CPU then writes to a device control register set up in the system memory for the expansion device that will transmit the data. When the CPU wants to notify the device of its pending task, it will do so through the PCI bridge. The device then fetches one or more of the descriptors in a single PCI transaction (PCI burst) and then generally one packet at a time until the entire data to be transmitted is fetched. The device then transmits the data as requested by the CPU.

PCI bridges may 'read ahead' of a transaction, or 'prefetch' data from the system memory with the idea being that having the data available to the device at the PCI bridge, rather than in the system memory, would speed the process. Unfortunately, the bridge does

not have a good estimate of how much data to prefetch. Bridges may end up prefetching too much data and having to discard the data. Prefetching of the data occupies the bus and the bridge, and wasting any data prefetched reduces the overall system efficiency.

This can lead to a high load on the PCI bus and the device, as well as slowing down the speed of transmission through the interface. It may also place a high load on the system memory, which in turn can slow down the effective speed of the CPU. This problem is compounded when another expansion bus is added to the system. When multiple busses exist in a system, there may be PCI bridges that bridge between the busses. These types of bridges are often referred to a PCI-to-PCI bridges, or P2P bridges. For ease of discussion, the term PCI bridge will be used to refer to both PCI and P2P bridges.

BRIEF DESCRIPTION OF THE DRAWINGS

Embodiments of the invention may be best understood by reading the disclosure with reference to the drawings, wherein:

Figure 1 shows an example of a system architecture employing a bridge.

Figure 2 shows a simplified block diagram of a system employing a bridge during an expansion bus transaction cycle.

Figure 3 shows a flow chart of an embodiment of a method for processing an expansion bus transaction.

Figure 4 shows a more detailed flow chart of an embodiment of a method to prefetch data.

DETAILED DESCRIPTION OF THE EMBODIMENTS

Figure 1 shows an embodiment of a system using an expansion bus. Devices, such as 18, reside on the expansion bus to add further functions and features to the system. The bridge device 14 provides communications between the system central processing unit 10, and device on the expansion bus 17. A bridge may reside on both the expansion bus and the local system bus 15, or it may reside between two expansion busses. The bridge may be a

digital signal processor, general-purpose processor, or an application specific integrated circuit (ASIC), as examples. All of these, as well as other examples, will be referred to here as 'processing elements.'

In either case, the bridge will have a port 144 to allow it to communicate on the expansion bus 17, and another port 142 to allow it to communicate on the system bus 15 or on another expansion bus, not shown. In one embodiment the system is a network device that employs the expansion device 18 as a network interface card. The central processing unit may have data it wishes to transmit across a network. This transmission operation may be the subject of a transaction between the CPU 10 and the device 18.

In general, in order to assist with the transaction, the bridge may institute a prefetching process to bring the data closer to the expansion device more quickly. Currently, however, the prefetching process relies on a prediction of how much data is needed, as there is no current means to communicate how much data is required for a transaction to the bridge devices. This results in adaptations of the prefetching process to overcome this lack of knowledge. One such process, set out in US Patent Application No. 10/742,185, (attorney docket no. 2705-306), uses a smart discard approach.

The smart discard approach is necessary because the bridge may prefetch data into the bridge and then have it become stale. Stale data is that data that does not reflect changes to the data made in the system memory. Stale data arises in part because the bridge does not know how much data to prefetch. A prefetch is generally the result of a read transaction from an expansion device.

An example of a transaction is shown in Figure 2. It must be noted that this diagram has been simplified for ease of discussion, and any ordering is merely for the same reason. The embodiments of the invention apply to other sequences as well. The CPU 10 is part of the system block that would include the system memory 12. The CPU 10 writes a series of descriptor blocks into a predetermined region of memory, such as region 120 of Figure 1.

These descriptor blocks describe the data to be transmitted, such as the addresses and size of each portion of data to be transmitted. The CPU then writes to the expansion device, requesting that the expansion device transmit the data across the network.

The bridge 14 then passes the write on to the expansion device 18, in this embodiment a network interface card, such as an Ethernet or other protocol interface card. The expansion device then issues a read request to fetch the descriptor blocks. The bridge passes this request onto the system, but also notes that the bridge should analyze the response from that portion of the memory. The bridge has knowledge of which portions of memory are used for descriptor blocks, so when the read request for addresses within that portion of memory passes through the bridge, the bridge identifies the request as one for which there may be a prefetch process needed.

The knowledge about the descriptor blocks used by a particular expansion device would typically be configured into the bridge when the device initializes. The software that allows the device to communicate with the outside world, the device driver, would configure the bridge with the information that would allow the bridge to recognize a read request for the descriptor blocks. The information may be the descriptor address space, offset of the packet length and the buffer address and descriptor size, and the ending address. Essentially, the necessary information is where the descriptors reside, and where in a descriptor block the bridge can find the length of the data to be transmitted, the length of the descriptor and the address of the particular packet to be transmitted.

When the descriptor blocks are read from system memory and pass through the bridge, the bridge would transmit them to the expansion device. In addition, the bridge parses the descriptor to locate the size of the packet to be transmitted, referred to here as the packet length or the transmit size, the location of the packet data, or the address of the data to be transmitted. This data that informs the bridge of the location and size of the data to be

operated upon by the device will be referred to here as the descriptor data. The bridge then stores the descriptor data in a table or other local memory on the bridge.

Storage in the table will probably involve storage in a hash table. A hash table uses shorter addresses, typically that last byte or two bytes of a full address. This allows for faster indexing of the data by the bridge to locate the desired data. If multiple descriptors are fetched, all the descriptor addresses are optimized and stored for faster access. The expansion device, having received its descriptor blocks, then issues a read request for the data.

The bridge now responds to that request by searching the hash table for the corresponding descriptor data using the address of the data in the read request as the key and determining the transmit size, then fetching the data requested as well as prefetching all of the necessary data for the expansion device. The bridge may prefetch the data based upon a read request or even before the read request.

For example, assume the packet length is 128 bytes and assume the device can only read 32 bytes at a time, which corresponds to "burst length" in PCI specification. When the device makes the request to read the first 32 bytes, the bridge can prefetch the entire 128 bytes. The bridge knows from the hash table the complete size of the packet. Requests for the remaining 3 sets of 32bytes of the packet from the device can be delivered by the bridge without going to the system memory as it as already prefetched the complete packet of 128 bytes.

The bridge could start prefetching before the first request, but this scheme will be a little more difficult to manage and the bridge will have to be more intelligent. Both schemes are possible and are included in the scope of this invention

Allowing the bridge to have the knowledge needed to prefetch the data needed, not more data which results in data discards, and not less data, which requires more reads to retrieve the needed data, increases the efficiency of the system. An embodiment of the process at the bridge is shown in flowchart form in Figure 3.

At 20, the bridge receives the read request from the expansion device. This is the read request triggered by the CPU write to the expansion device. At 22 the read request is issued to the portion of system memory predetermined to have the descriptor addresses. At 24, the descriptor blocks including the descriptor data are received at the bridge. The bridge parses the descriptor data to identify the size of the data transmission at 26.

The descriptor data is then stored in the hash table memory at 28. If the memory is full at 36, which is unlikely but could happen on a high volume, multi-channel device, the oldest descriptor is discarded and the space reused for the new descriptor at 38. Another advantage of this process is that the table in which the descriptor data is stored only has to be accessed once. It will then be freed up for other devices or processes to access it.

In an alternative to the discarding of the oldest entry, the bridge may track the status of the data. The bridge may be able to determine if the device has consumed all of the data associated with a particular descriptor. If the data has been consumed, it can either be marked as 'used' data, or flushed.

After fetching the descriptor, the expansion device tries to read the packet from the system memory through the bridge in order to transmit it out at 30. The bridge scans the address to see if it falls in the descriptor address space at 31. If it does, the process returns to 22. Otherwise, the bridge scans the hash table for that address.

If no match is found in the hash table, then the bridge behaves like a standard bridge and prefetches data based on cacheline size and the PCI command. If a match is found, the bridge knows that the device is trying to access a packet for transmission. From the hash table the bridge knows the packet size and prefetches the whole packet. At 32, the bridge begins prefetching the data from the system memory. The bridge knowing the transmit size has several benefits. Once the bridge has all of the data it needs for a particular prefetch process, the bridge can disconnect from the system bus at 34, or the expansion bus on the system bus

side of the bridge, and remain connected to the expansion bus upon which the expansion device resides. This decreases the load on the system, or system-side bus.

Currently, if a bridge close to the CPU breaks a transaction before the complete packet is transferred, the device has to reinitiate a request for the remaining data. In the embodiments of the invention, the bridge knows exactly how much data in which the device is interested, and can prefetch the remaining data without a request from the device shown at 39. When the device reconnects and reinitiates the request, the bridge can handle the request locally, avoiding the overhead and delay involved in going back to the CPU.

As mentioned above, the prefetching of the data may have several parts to the process. When the read request comes into the bridge, the bridge accesses the descriptor data from the table and determines the transmit data size at 40 in Figure 4. Also as mentioned above, this information may be in the form of an offset length into the descriptor data at which the packet length is located.

At 42 a read request for a particular size is issued to the system memory. In PCI systems, the read request may be a memory read (MR), a memory read line (MRL) or a memory read multiple line (MRM). An MR is typically 4 bytes of data, also the typical length of the descriptor blocks, and an MRL is for a cache line size, where the cache line size is configured when the bridge is initialized. An MRM is for some multiple of the cache line size. When the data of whichever size arrives back at the bridge, the data is then transmitted to the expansion device at 44.

In this manner, the expansion device receives the data it requires, such as that to be transmitted across the network, with minimal overhead on the system bus. In addition, the data is acquired with little waste in the prefetching process. The prefetching process reduces the load on the system bus. The embodiments of a prefetching process as set out here avoid the waste and inefficiency that can come with some prefetching processes as discussed above.

Thus, although there has been described to this point a particular embodiment for a method and apparatus for improving efficiency in systems using expansion busses and bridges, it is not intended that such specific references be considered as limitations upon the scope of this invention except in-so-far as set forth in the following claims.